**Java masterclass**

**Particularidades da linguagem e convenções:**

1. O Java é fortemente tipado, sendo necessário declarar qual o tipo de variável, retorno de funções e métodos e o tipo de dado nos parâmetros de uma função.
2. Para a declaração de classes, utilizar PascalCase, e, para a declaração de métodos, utilizar camelCase.

**Boas práticas:**

1. Métodos devem ter no máximo três parâmetros, e devem ser claros no que fazem, e particionados de modo que façam apenas uma única função.

**Comentários:**

Comentários explicam o que partes do código fazem, e ajudam os programadores a entender o código em si como um todo. Em java, os comentários podem ser bastante descritivos, utilizando-se técnicas únicas, que, junto com a IDE, podem resultar em algo bem explicativo, como por exemplo, a utilização de **Javadoc comments** (comentários de documentação). É possível incluir até HTML em um javadoc comment:

1. /\*\*

2. \* Represents a person with name and age attributes.

3. \*/

4. public class Person {

5. private String name;

6. private int age;

7.

8. /\*\*

9. \* Constructs a person with default values of name = "Unknown" and age = 0.

10. \*/

11. public Person() {

12. this.name = "Unknown";

13. this.age = 0;

14. }

15.

16. /\*\*

17. \* Constructs a person with the specified name and age.

18. \* @param name the name of the person

19. \* @param age the age of the person

20. \*/

21. public Person(String name, int age) {

22. this.name = name;

23. this.age = age;

24. }

25.

26. /\*\*

27. \* Returns the name of the person.

28. \* @return the name of the person

29. \*/

30. public String getName() {

31. return name;

32. }

33.

34. /\*\*

35. \* Sets the name of the person.

36. \* @param name the name of the person

37. \*/

38. public void setName(String name) {

39. this.name = name;

40. }

41.

42. /\*\*

43. \* Returns the age of the person.

44. \* @return the age of the person

45. \*/

46. public int getAge() {

47. return age;

48. }

49.

50. /\*\*

51. \* Sets the age of the person.

52. \* @param age the age of the person

53. \*/

54. public void setAge(int age) {

55. this.age = age;

56. }

57. }

58.

**Classes (POO):**

**Classes e Objetos:**

Classes são moldes de objetos, estes últimos que em Java, são criados com a palavra-chave **new** a partir de uma classe. Quando um objeto é criado, um local na memória exclusivo para ele é criado.

1. public class Main {

2. public static void main(String[] args) {

3. Person person1 = new Person(25, "John");

4. Person person2 = new Person(30, "Emily");

5. }

6. }

7.

8. public class Person {

9. private int age;

10. private String name;

11.

12. public Person(int age, String name) {

13. this.age = age;

14. this.name = name;

15. }

16.

17. public int getAge() {

18. return age;

19. }

20.

21. public void setAge(int age) {

22. this.age = age;

23. }

24.

25. public String getName() {

26. return name;

27. }

28.

29. public void setName(String name) {

30. this.name = name;

31. }

32. }

33.

**Constructors:**

Os construtores (constructors), são métodos que sempre são chamados quando uma classe é inicializada (instanciada em objetos), e, podem obrigar a quem as utiliza a fornecer parâmetros iniciais ao instanciar uma classe, ou não.

1. public class Main {

2. public static void main(String[] args) {

3. Person anonymousPerson = new Person(25);

4. Person john = new Person("John", 25);

5. }

6. }

7.

8. public class Person {

9. private String name;

10. private int age;

11.

12. // Constructor for anonymous people

13. public Person(int age) {

14. // This calls the constructor with name and age parameters, coded below

15. this("Anonymous", age);

16. }

17.

18. // Constructor with name and age parameters

19. public Person(String name, int age) {

20. this.name = name;

21. this.age = age;

22. }

23.

24. // Getter for name

25. public String getName() {

26. return name;

27. }

28.

29. // Getter for age

30. public int getAge() {

31. return age;

32. }

33.

34. // Setter for name

35. public void setName(String name) {

36. this.name = name;

37. }

38.

39. // Setter for age

40. public void setAge(int age) {

41. this.age = age;

42. }

43. }

44.

No exemplo acima, ocorre um **overload** de constructors.

**Abstract classes:**

As **classes abstratas** são classes inicializadas com a palavra-chave **abstract**, e cuja função é servir de modelo, ou protótipo de herança para outras classes. Classes abstratas não podem ser instanciadas em objetos diretamente, servem apenas para serem herdadas. Elas podem conter métodos abstratos, os quais devem ser sobrescritos na classe que a herda (**@Override**), implementando uma função específica. Classes abstratas podem ter constructors, o qual pode ser chamado na classe que a herda através do método **super()**. Classes abstratas podem guardar dados.

**Interfaces:**

As **interfaces** são também modelos (protótipos), que podem ser usados múltiplas vezes em classes, através da palavra-chave **implements**, e, diferente de classes abstratas, elas não podem guardar dados a princípio (exceção a campos estáticos).